Overview: In this lab, you will work in groups of three, experimenting in the use of git (for your local version of a repository) and GitHub (where, via git, the three of you will combine your work). You will ultimately implement a working solution to a collection of C++ classes/programs that includes:

```
IntegerVector.{h,cpp} CharacterVector.{h,cpp} DoubleVector.{h,cpp} VectorTester.cpp
```

Each class depends on the other two classes, and `VectorTester.cpp` will eventually test all three classes.

Groups: You have been assigned at random into a group of three as shown below, with your associated GitHub repository listed:

<table>
<thead>
<tr>
<th>team</th>
<th>netids</th>
<th>repository URL</th>
</tr>
</thead>
<tbody>
<tr>
<td>blue</td>
<td>mc8na, tl3fx, ws7rq</td>
<td><a href="https://github.com/barrylawson/cmsc240_s2018_lab7_blue">https://github.com/barrylawson/cmsc240_s2018_lab7_blue</a></td>
</tr>
<tr>
<td>cyan</td>
<td>wr5gk, ee6gj, jm6ms</td>
<td><a href="https://github.com/barrylawson/cmsc240_s2018_lab7_cyan">https://github.com/barrylawson/cmsc240_s2018_lab7_cyan</a></td>
</tr>
<tr>
<td>gold</td>
<td>ew5jm, hn3gn, as9kc</td>
<td><a href="https://github.com/barrylawson/cmsc240_s2018_lab7_gold">https://github.com/barrylawson/cmsc240_s2018_lab7_gold</a></td>
</tr>
<tr>
<td>green</td>
<td>ag6bi, mx2ww, pt2mv</td>
<td><a href="https://github.com/barrylawson/cmsc240_s2018_lab7_green">https://github.com/barrylawson/cmsc240_s2018_lab7_green</a></td>
</tr>
<tr>
<td>navy</td>
<td>ws8te, lc5up, tl3xd</td>
<td><a href="https://github.com/barrylawson/cmsc240_s2018_lab7_navy">https://github.com/barrylawson/cmsc240_s2018_lab7_navy</a></td>
</tr>
<tr>
<td>orange</td>
<td>mg4fp, jo6kg, vc5xa</td>
<td><a href="https://github.com/barrylawson/cmsc240_s2018_lab7_orange">https://github.com/barrylawson/cmsc240_s2018_lab7_orange</a></td>
</tr>
<tr>
<td>purple</td>
<td>he7is, ch2gy, ba5wp</td>
<td><a href="https://github.com/barrylawson/cmsc240_s2018_lab7_purple">https://github.com/barrylawson/cmsc240_s2018_lab7_purple</a></td>
</tr>
<tr>
<td>red</td>
<td>rh7fi, hz2bz</td>
<td><a href="https://github.com/barrylawson/cmsc240_s2018_lab7_red">https://github.com/barrylawson/cmsc240_s2018_lab7_red</a></td>
</tr>
</tbody>
</table>

Getting Started:

- Begin by remotely logging in to one of the `mathcs*` or `turing2` machines, and then create a new `lab7` directory within your own directory structure for this course. Then change into your `lab7` directory.
- Begin by cloning the original version of your repository, using your corresponding URL given in the table above:
  ```
git clone https://github.com/barrylawson/cmsc240_s2018_lab7_{YOUR TEAM}
```
- Change into the resulting directory.

Process:

- Using the commands below, you will show the current local git branch(es), create a new branch (using your netid), and change to that branch:
  ```
git branch
$ git branch {YOUR NETID}
git branch
$ git checkout {YOUR NETID}
git branch
```
• Now do your work on your own non-master branch. Specifically:
  
  – In the two .cpp files that are not assigned to you, add code to the non-void stubs sufficient to make them compile (just return 0).
  – Add the appropriate code to the methods in your assigned .cpp file.
  – Add code in the appropriate spot in VectorTester.cpp to fully test your class’s implementation. (Do not yet add tester code in the “appended-to” sections.) Because one of the classes relies on characters, make sure that you are inserting integer/double/character values that correspond to printable ASCII characters (see https://www.ascii-code.com/).
  – Use make to compile your code, and fix any errors, repeating until your implementation is correct.

• Commit your work on your non-master branch. It is good practice to regularly check the status of your commits and to regularly pull from the remote (GitHub) master branch to pull in any updates others have made. (If you receive conflict issues when trying to pull, see the “Handling Pull Conflicts” section below.)

```
  git status
  git pull origin master  # (updates your local master and your branch)
  git commit -a -m {MESSAGE INDICATING YOUR CHANGES}
```

• Now switch back to the master branch, then merge your non-master branch to the master. (By using cat, you’ll see that your updates aren’t seen in the master branch until you merge.)

```
  git branch
  git checkout master
  git branch
  cat {YOUR CLASS}.cpp
  git merge {YOUR NETID}
  cat {YOUR CLASS}.cpp
```

• Now push your changes to the shared repository on GitHub:

```
  git status
  git push origin master
  git remote show origin
```

The repository on GitHub should now reflect the changes seen on your local repositories. Visit your GitHub repository in a browser and you should see your (and perhaps others’) code changes.

• Now continue to do work on your non-master branch, adding the additional “appended-to” tests in VectorTester.cpp corresponding to your assigned class. Merge your non-master branch into your master, push, and repeat the process until your team has a fully-implemented version of all classes and the tester — on GitHub and in each of your local repositories.

```
  git checkout {YOUR NETID}
  git branch
  // do work in VectorTester.cpp
  git status
  git pull origin master
  git commit -a
```

```
  git checkout master
  git branch
  git merge {YOUR NETID}
  git status
  git push origin master
  git remote show origin
```
**Handling Pull Conflicts:** When doing a fetch-and-merge from the remote repository (via `git pull origin master`), git will do its best to merge your local repository's code with (potentially different) code that exists on the remote repository. In many cases — particularly if you make a practice of regularly keeping your local repo up-to-date via regular pulls — git can handle merging itself using the “recursive strategy”. In such cases, you should see a message similar to the following:

```
Merge made by the 'recursive' strategy.
```

In some cases, however, git's recursive strategy will not work, and conflicts will result that you must handle explicitly. (One such situation is a conflict of a teammate's “good” code in their assigned class versus your “stub” code in their assigned class.) When git cannot automatically merge, you will see a message similar to the following:

```
% git pull origin master
From https://github.com/barrylawson/cmse240_2018lab7_orange
    * branch     master -> FETCH_HEAD
Auto-merging VectorTester.cpp
Auto-merging IntegerVector.cpp
CONFLICT (content): Merge conflict in IntegerVector.cpp
Auto-merging DoubleVector.cpp
CONFLICT (content): Merge conflict in DoubleVector.cpp
Auto-merging CharacterVector.cpp
CONFLICT (content): Merge conflict in CharacterVector.cpp
Automatic merge failed; fix conflicts and then commit the result.
```

You will then need to edit each file for which there is a conflict:

- Remove any of the following lines added by git:

  ```
  >>>>>>> HEAD
  =======
  >>>>>>> (hash)
  ```

- Delete any incorrect lines of code (e.g., old stubs).
- Keep the correct lines of code.

Then commit your code on your non-master branch, merge into your master branch, and then push:

```
git commit -a       # (will offer a merge comment in vim – just :wq to accept)
git branch
git checkout master
git branch
git merge {YOUR_NETID}
git push origin master
git remote show origin
```

---

**Submitting:** After all team members have pushed all their final changes, each person should do a final pull from the origin master. In this way, you should all have an (otherwise) identical local master branch within your own directory.

Package your `lab7` directory (containing your local repo) into a gzipped tarball a la `cmse240_1ab7_{netid}.tgz` and drop into the `lab7/` folder of your shared Box folder. Your lab is due by 23:59:59 on Fri 09 Mar.